Vol.2016-HPC-154 No.9
2016/4/25

中心-半径型区間演算に基づく矩形演算を用いた精度保証付き高速フーリエ変換

篠塚 敬介[1,a)]　高橋 大介[2,b)]

概要：数値計算で生じる誤差の範囲は精度保証付き数値計算を行うことによって正確に見積もることができる．実数の計算では，上限-下限型区間演算および中心-半径型区間演算が，複素数の計算では，上限-下限型区間演算に基づく矩形演算および円板演算が，それぞれ精度保証法として知られている．ここで，精度保証の対象となる数値計算のアルゴリズムの一つに高速フーリエ変換（fast Fourier transform，FFT）があり，複素数の計算における精度保証法が適用されてきた．本稿では，既存手法よりも高速に計算することを目的とした中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT を提案し，その性能評価の結果について述べる．

1. はじめに

数値計算は数学上の問題を計算機で解く手法であり，現在に至るまで科学・工学の分野で重要な役割を果たしている．しかし，数値計算は計算機内部での実数の表現が有限桁の近似計算で行われるため，正しい計算結果が得られないことがある．そこで，計算結果の誤差評価を行うことで正しい計算結果との誤差の範囲を正確に見積もる精度保証付き数値計算が研究されてきた [1]．精度保証付き数値計算の手法として，実数の計算では，上限-下限型区間演算および中心-半径型区間演算が知られている [2]．また，複素数の計算では，上限-下限型区間演算に基づく矩形演算および円板演算が知られている [3]．

数値計算で頻出するアルゴリズムに，離散フーリエ変換（discrete Fourier transform，DFT）を高速に計算する高速フーリエ変換（fast Fourier transform，FFT）がある [4]．近年，野村は精度保証付き FFT を提案した [5]．FFT は複素数の計算であることから，提案された精度保証付き FFT は，上限-下限型区間演算に基づく矩形演算および円板演算を用いている．しかし，野村による Mathematica [6] 上での実験では，浮動小数点演算による FFT と比較して，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT の計算時間が 2 から 1024 までのデータ点数において最大で約 20 倍，円板演算を用いた精度保証付き FFT の計算時間が最大で約 50 倍となり，精度保証に多くの時間を要していた．また，Rump は MATLAB [7] や Octave [8] 上で精度保証付き数値計算を行うソフトウェアである INTLAB [9] で円板演算を用いた精度保証付き FFT の実装を行った．一方，Chonali は 8 点の精度保証付き FFT を上限-下限型区間演算に基づく矩形演算を用いて FPGA 上で実装した [10]．

ここで，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT は提案されているが，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT は提案されていない．そこで，本稿では，より高速な精度保証付き FFT の実現を目的として中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT を提案する．

[1] 筑波大学大学院システム情報工学研究科
[2] 筑波大学システム情報系
[a)] ksksnzk@hpcs.cs.tsukuba.ac.jp
[b)] daisuke@cs.tsukuba.ac.jp

2. 精度保証付き数値計算の基本的事項

2.1 浮動小数点数

浮動小数点数のフォーマットは，多くの計算機において IEEE 標準 754 で規定されている [1]．IEEE754 に基づく浮動小数点数の集合を $\mathbb{F}$ で表す．$\mathbb{F}$ における演算では，4 つの丸めモードが指定できる [1]．なお，c を実数 ($c \in \mathbb{R}$) とする．

上向きの丸め

c 以上の浮動小数点数の中で最も小さい数に丸める．これを $\triangle : \mathbb{R} \to \mathbb{F}$ と表す．

下向きの丸め

c 以下の浮動小数点数の中で最も大きい数に丸める．これを $\nabla : \mathbb{R} \to \mathbb{F}$ と表す．

最近点への丸め

c に最も近い浮動小数点数に丸める．これを $\square : \mathbb{R} \to \mathbb{F}$ と表す．もし，このような点が 2 点ある場合には，仮

Algorithm 1 上限-下限型区間演算における四則演算 [5]

$$
\begin{aligned}
[a] + [b] &= [\nabla(a_1 + b_1),\ \triangle(a_2 + b_2)] \\
[a] - [b] &= [\nabla(a_1 - b_2),\ \triangle(a_2 - b_1)] \\
[a] \cdot [b] &= [\min\{\nabla a_1 b_1,\ \nabla a_1 b_2,\ \nabla a_2 b_1,\ \nabla a_2 b_2\}, \\
&\qquad \max\{\triangle a_1 b_1,\ \triangle a_1 b_2,\ \triangle a_2 b_1,\ \triangle a_2 b_2\}] \\
[a] \,/\, [b] &= [\min\{\nabla a_1 / b_1,\ \nabla a_1 / b_2,\ \nabla a_2 / b_1,\ \nabla a_2 / b_2\}, \\
&\qquad \max\{\triangle a_1 / b_1,\ \triangle a_1 / b_2,\ \triangle a_2 / b_1,\ \triangle a_2 / b_2\}]
\end{aligned}
$$

数部の最後のビットが偶数である浮動小数点数に丸める．これを偶数丸め方式という．

切り捨て

絶対値が c 以下の浮動小数点数の中で c に最も近いものに丸める．

2.2 区間演算

区間演算は実数を区間で包含し，その区間を単位とした四則演算を行う．ここで，区間は「上限」と「下限」の 2 つの数で表現したものと「中心」と「半径」の 2 つの数で表現したものが知られている．本稿では，「上限」と「下限」で区間を表現した区間演算を上限-下限型区間演算と表記する．また，「中心」と「半径」で区間を表現した区間演算を中心-半径型区間演算と表記する．

計算機上で区間演算を実現する場合，区間を表現する数を浮動小数点数とし，IEEE754 に基づく丸めモード変更を利用する．これを，機械区間演算 [1] と呼ぶ．本節以降，特に断らない限り，区間演算と述べた場合には機械区間演算を想定する．

2.2.1 上限-下限型区間演算

上限-下限型区間演算では，「上限」と「下限」の 2 つの数を浮動小数点数とし，区間を式 (1) によって表現する．ただし，$a_1, a_2 \in \mathbb{F}$ である．

$$[a] = [a_1,\ a_2] = \{x \in \mathbb{R} \colon a_1 \le x \le a_2\} \tag{1}$$

上限-下限型区間演算における四則演算は Algorithm 1 のようになる [5].

2.2.2 中心-半径型区間演算

中心-半径型区間演算では，「中心」と「半径」の 2 つの数を浮動小数点数とし，区間を式 (2) によって表現する．ただし，$a, \alpha \in \mathbb{F}$ である．

$$\langle a \rangle = \langle a,\ \alpha \rangle = \{x \in \mathbb{R} \colon a - \alpha \le x \le a + \alpha\} \tag{2}$$

中心-半径型区間演算における四則演算のうち，加算・減算・乗算については Algorithm 2 のようになる [11]．また，$\epsilon' = 2^{-53}$ は 1.0 未満の最大の浮動小数点数と 1.0 の差として定数であり，$\eta = 2^{-1074}$ は最小の正の非正規化浮動小数点数である [11].

ここで, 除算に関しては, 最初に $1/\langle b \rangle$ の計算を行い, 次に $\langle a \rangle$ との乗算を行えばよい．Algorithm 3 に中心-半径型区間演算における $1/\langle b \rangle$ の計算方法を示す [11]．Algorithm 4 に中心-半径型区間演算における除算の計算方法を示す [11]．ただし，$1/\langle b \rangle = \langle c' \rangle = \langle c',\ \gamma' \rangle$ とする．

また，上限-下限型区間から中心-半径型区間への変換に関して，Algorithm 5 が知られている [1]．ただし，$[a_1,\ a_2]$ から $\langle m,\ r \rangle$ に変換するものとする．このとき，式 (3) が成立する．

$$[a_1,\ a_2] \subset \langle m,\ r \rangle \tag{3}$$

中心-半径型区間演算は上限-下限型区間演算と比較して，積の計算を条件分岐を行わずにシンプルに計算することができるが，区間幅が広がる過大評価が起こる．ただし，過大評価は上限-下限型区間演算の正確な計算によって得られる区間幅の 1.5 倍以下に抑えられることが知られている [11].

Algorithm 2 中心-半径型区間演算における加算・減算・乗算 [11]

$$
\begin{aligned}
\langle a \rangle + \langle b \rangle &= \langle c = \square\,(a + b),\ \triangle\,(\epsilon' \cdot |c| + \alpha + \beta) \rangle \\
\langle a \rangle - \langle b \rangle &= \langle c = \square\,(a - b),\ \triangle\,(\epsilon' \cdot |c| + \alpha + \beta) \rangle \\
\langle a \rangle \cdot \langle b \rangle &= \langle c = \square\,(a \cdot b),\ \triangle\,\{\eta + \epsilon' \cdot |c| + (|a| + \alpha)\beta + \alpha|b|\} \rangle
\end{aligned}
$$

Algorithm 3 中心-半径型区間演算における $1/\langle b \rangle$ の計算 [11]

$$
\begin{aligned}
c'_1 &= \nabla\,\{(-1)/(-|b| - \beta)\} \\
c'_2 &= \triangle\,\{(-1)/(-|b| + \beta)\} \\
c' &= \triangle\,\{c'_1 + 0.5 \cdot (c'_2 - c'_1)\} \\
\gamma' &= \triangle\,(c' - c'_1) \\
c' &= \mathtt{sign}(b) \cdot c'
\end{aligned}
$$

Algorithm 4 中心-半径型区間演算における除算 [11]

$$
\begin{aligned}
\langle a \rangle / \langle b \rangle &= \langle a \rangle \cdot \langle c' \rangle \\
&= \langle c = \square\,(a \cdot c'),\ \triangle\,\{\eta + \epsilon' \cdot |c| + (|a| + \alpha)\,\gamma' + \alpha|c'|\} \rangle
\end{aligned}
$$

Algorithm 5 上限-下限型区間から中心-半径型区間への変換 [1]

上向きの丸めモードに変更

$$
\begin{aligned}
m &= \frac{a_2 - a_1}{2} + a_1 \\
r &= m - a_1
\end{aligned}
$$

3. 既存の精度保証付き高速フーリエ変換

本章では，まず，FFT の基となる DFT について 3.1 節で説明する．そして，FFT について 3.2 節で説明し，既存の精度保証付き FFT を 3.3 節で説明する．

3.1 離散フーリエ変換

DFT は連続フーリエ変換において無限区間積分を有限の和で書き換え，周波数領域と時間領域をともに離散化したものである．n 点のデータに対する DFT は式 (4) で定義される [4]．ただし，$x(j)$ および $y(k)$ は複素数であり，$\omega_n = e^{-2\pi i/n}$，$i = \sqrt{-1}$，$0 \le k \le n-1$ である．

$$y(k) = \sum_{j=0}^{n-1} x(j)\,\omega_n^{jk} \tag{4}$$

3.2 高速フーリエ変換

FFT は DFT を高速に計算するためのアルゴリズムである．n が 2 で割り切れる場合，式 (4) は $n/2$ 点の前半部分と $n/2$ 点の後半部分に分けることができ，式 (5) のように書くことができる．ただし，$0 \le k \le n-1$ である．

$$y(k) = \sum_{j=0}^{n/2-1} \{x(j) + x(j+n/2)\,\omega_n^{(n/2)k}\}\,\omega_n^{jk} \tag{5}$$

ここで，$\omega_n = e^{-2\pi i/n}$ より，

$$\omega_n^{(n/2)k} = e^{-\pi ik} = \begin{cases} 1 & k \text{ が偶数} \\ -1 & k \text{ が奇数} \end{cases} \tag{6}$$

となる．

したがって，k が偶数と奇数の場合を考えると，n 点 DFT は式 (7)，(8) の 2 つの $n/2$ 点 DFT に分解することができる．ただし，$0 \le k \le n/2-1$ である．

$$y(2k) = \sum_{j=0}^{n/2-1} \{x(j) + x(j+n/2)\}\,\omega_{n/2}^{jk} \tag{7}$$

$$y(2k+1) = \sum_{j=0}^{n/2-1} \{\{x(j) - x(j+n/2)\}\,\omega_n^{j}\}\,\omega_{n/2}^{jk} \tag{8}$$

n が 2 のべき乗であるとき，FFT は上記の式 (7)，(8) の分解を再帰的に行い，2 点 DFT に帰着させることで実現できる．

ここで，FFT は入力データが出力データに上書きされる in-place アルゴリズムである Cooley-Tukey アルゴリズムが知られている．一方，入力データと出力データを別の配列にする out-of-place な Stockham アルゴリズムが知られている [12]．Stockham アルゴリズムはビットリバース順の並び替えが必要なく，ベクトル化に向いている [13]．そこで，本稿では Stockham アルゴリズムを考える．

3.3 上限-下限型区間演算に基づく矩形演算による精度保証付き高速フーリエ変換

矩形演算は複素数を複素矩形領域で包含し，その複素矩形領域を単位とした四則演算を行う．ここで，複素矩形領域は「実部」と「虚部」で表現され，「実部」と「虚部」は上限-下限型区間で表現される．本稿では上限-下限型区間で「実部」と「虚部」を表現した矩形演算を上限-下限型区間演算に基づく矩形演算と呼ぶこととする．

Algorithm 6 上限-下限型区間演算に基づく矩形演算における四則演算 [5]

$$\begin{aligned}
[[a_R],[a_I]] + [[b_R],[b_I]] &= [[a_R]+[b_R],[a_I]+[b_I]] \\
[[a_R],[a_I]] - [[b_R],[b_I]] &= [[a_R]-[b_R],[a_I]-[b_I]] \\
[[a_R],[a_I]] \cdot [[b_R],[b_I]] &= [[a_R][b_R]-[a_I][b_I],[a_R][b_I]+[a_I][b_R]] \\
[[a_R],[a_I]] \,/\, [[b_R],[b_I]] &= \left[\frac{[a_R][b_R]+[a_I][b_I]}{[b_R]^2+[b_I]^2}, \frac{-[a_R][b_I]+[a_I][b_R]}{[b_R]^2+[b_I]^2}\right]
\end{aligned}$$

上限-下限型区間演算に基づく矩形演算における複素矩形領域を式 (9) に示す．ただし，$[a_R] = [a_{R1}, a_{R2}], [a_I] = [a_{I1}, a_{I2}]$ であり，$a_{R1}, a_{R2}, a_{I1}, a_{I2} \in \mathbb{F}$ である．

$$[[a_R],[a_I]] = \{x + iy \colon x \in [a_R], y \in [a_I]\} \tag{9}$$

上限-下限型区間演算に基づく矩形演算における四則演算は Algorithm 6 のようになる [5]．

野村は上限-下限型区間演算に基づく矩形演算を用いて FFT の複素数の計算における精度保証を行い，Mathematica 上での実験結果では，浮動小数点演算による FFT と比較して，2 から 1024 までのデータ点数において計算時間が最大で約 20 倍となった [5]．一方，野村は複素数を複素円板領域で包含した円板演算によっても FFT の精度保証を行ったが，Mathematica 上での実験結果では，浮動小数点演算による FFT と比較して，2 から 1024 までのデータ点数において計算時間が最大で約 50 倍となった [5]．ただし，データ点数が 2 から 128 と小さいとき，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT は円板演算を用いた精度保証付き FFT よりも最大区間幅が小さく，データ点数が 256 から 1024 と大きいとき，円板演算を用いた精度保証付き FFT は上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT よりも最大区間幅が大きくなった [5]．本稿では，より高速に計算をすることができた矩形演算に着目する．

4. 提案する精度保証付き高速フーリエ変換

本章では，まず，提案する精度保証付き FFT を 4.1 節で述べる．さらに，本稿における実装上の工夫を 4.2 節，4.3 節，4.4 節，4.5 節でそれぞれ述べる．

4.1 中心-半径型区間演算に基づく矩形演算による精度保証付き高速フーリエ変換

3.3 節において，複素矩形領域は「実部」と「虚部」で表現され，「実部」と「虚部」は上限-下限型区間で表現されていたが，本稿では，「実部」と「虚部」を中心-半径型区間で表現することを提案する．そして，中心-半径型区間で

Algorithm 7 中心-半径型区間演算に基づく矩形演算における四則演算

$$[\langle a_R\rangle,\langle a_I\rangle]+[\langle b_R\rangle,\langle b_I\rangle] = [\langle a_R\rangle+\langle b_R\rangle,\langle a_I\rangle+\langle b_I\rangle]$$
$$[\langle a_R\rangle,\langle a_I\rangle]-[\langle b_R\rangle,\langle b_I\rangle] = [\langle a_R\rangle-\langle b_R\rangle,\langle a_I\rangle-\langle b_I\rangle]$$
$$[\langle a_R\rangle,\langle a_I\rangle]\cdot[\langle b_R\rangle,\langle b_I\rangle] = [\langle a_R\rangle\langle b_R\rangle-\langle a_I\rangle\langle b_I\rangle,\langle a_R\rangle\langle b_I\rangle+\langle a_I\rangle\langle b_R\rangle]$$
$$[\langle a_R\rangle,\langle a_I\rangle]\,/\,[\langle b_R\rangle,\langle b_I\rangle] = \left[\frac{\langle a_R\rangle\langle b_R\rangle+\langle a_I\rangle\langle b_I\rangle}{\langle b_R\rangle^2+\langle b_I\rangle^2},\frac{-\langle a_R\rangle\langle b_I\rangle+\langle a_I\rangle\langle b_R\rangle}{\langle b_R\rangle^2+\langle b_I\rangle^2}\right]$$

```
1 Down()
2 x1 ← a1 + b1
3 Up()
4 x2 ← a2 + b2
5
6 Down()
7 y1 ← c1 - d2
8 Up()
9 y2 ← c2 - d1
```

図 1 上限-下限型区間演算で和と差の計算を行うプログラムにおける丸めモード変更回数削減前

```
1 Down()
2 x1 ← a1 + b1
3 y1 ← c1 - d2
4
5 Up()
6 x2 ← a2 + b2
7 y2 ← c2 - d1
```

図 2 上限-下限型区間演算で和と差の計算を行うプログラムにおける丸めモード変更回数削減後

「実部」と「虚部」を表現した矩形演算を中心-半径型区間演算に基づく矩形演算と呼ぶこととする．

中心-半径型区間演算に基づく矩形演算における複素矩形領域を式 (10) に示す．ただし，$\langle a_R\rangle = \langle a_R,\alpha_R\rangle$, $\langle a_I\rangle = \langle a_I,\alpha_I\rangle$ であり，$a_R,\alpha_R,a_I,\alpha_I \in \mathbb{F}$ である．

$$[\langle a_R\rangle,\langle a_I\rangle] = \{x+iy : x\in\langle a_R\rangle, y\in\langle a_I\rangle\} \tag{10}$$

中心-半径型区間演算に基づく矩形演算における四則演算は Algorithm 7 のようになる．

精度保証付き FFT は 3.3 節と同様に中心-半径型区間演算に基づく矩形演算を用いて FFT の複素数の計算を行えばよい．

4.2 丸めモード変更の回数削減

本節までに示した各区間演算および各矩形演算を用いた精度保証付き数値計算は計算順序を工夫することにより，丸めモード変更の回数を削減できることがある．例えば，実数 a,b の和を実数 x に，実数 c,d の差を実数 y に代入するプログラムを考える．これを上限-下限型区間演算を用いて精度保証する場合，$[x1,x2] = [a1,a2]+[b1,b2]$ と $[y1,y2] = [c1,c2]-[d1,d2]$ の計算を行えばよい．このとき，Algorithm 1 を単位として計算すると図 1 のようになる．ただし，`Down()` は下向きの丸めモードに変更する関数を，`Up()` は上向きの丸めモードに変更する関数をそれぞれ表す．ここで，図 1 では，丸めモード変更を行う関数を計 4 回呼び出しているが，丸めモード変更は `Down()` と `Up()` の 2 種類のみである．したがって，`Down()` の計算を先に行い，`Up()` の計算を後に行うことで丸めモード変更回数を 2 回に削減することが考えられる．実際，今回の例では和と差の各演算は順序を変更しても計算結果に影響しないため，丸めモード変更回数は図 2 のように 2 回に削減できる．中心-半径型区間演算や上限-下限型区間演算に基づく矩形演算，中心-半径型区間演算に基づく矩形演算も同様にして丸めモード変更の回数を削減できることがある．

本稿が対象とする FFT は最内側ループに複素数の計算が現れる．上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT において，最内側ループにおける複素数の計算を上限-下限型区間演算に基づく矩形演算を用いて行うが，このとき，丸めモード変更は計 20 回行われる．ここで，先述した考え方と同様にして計算結果に影響を与えない範囲で計算順序を変えると，最内側ループでの丸めモード変更回数が計 7 回に削減できる．

一方，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT において，最内側ループにおける複素数の計算を中心-半径型区間演算に基づく矩形演算を用いて行うが，このとき，丸めモード変更は計 20 回行われる．ここで，先述した考え方と同様にして計算結果に影響を与えない範囲で計算順序を変えると，最内側ループでの丸めモード変更回数が計 2 回に削減できる．

4.3 ベクトル化

丸めモード変更を実行する処理を入れるとベクトル化を行うことが困難になる．ここで，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の最内側ループでは丸めモード変更の回数が最近点への丸めを行う `Near()` と `Up()` の計 2 回である．最内側ループを `Near()` のモードで計算を行うループと `Up()` のモードで計算を行うループの 2 つに分割する．このとき，各ループには丸めモード変更を行う関数は 1 つだけとなるため，1 つ目の最内側ループの外に `Near()` を出すことができ，2 つ目の最内側ループの外に `Up()` を出すことができる．したがって，各ループはベクトル化を行うことができるようになる．本稿では，以上に述べた方法で中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の最内側ループにおけるベクトル化を行う．ただし，1 つ目のループで使用する

表 1 Intel AVX2 命令における主な融合積和演算命令

命令	説明
VFMADD132PD	Fused Multiply-Add of
VFMADD213PD	Packed Double-Precision
VFMADD231PD	Floating-Point Values
VFMADD132SD	Fused Multiply-Add of
VFMADD213SD	Scalar Double-Precision
VFMADD231SD	Floating-Point Values

変数の値を 2 つ目のループで使用する変数に対応させるため，ループ内の各変数を配列に変更することに注意する．また，配列に変更したことによる速度低下をできるだけ防ぐため，最内側ループでは大きなループを小さなループに分けるストリップマイニングを用いる．

4.4 融合積和演算の利用

融合積和演算は $x \times y + z$ で表される積和演算を 1 回の丸めで行う演算である．ここで，以降は Intel x86_64 アーキテクチャに基づく CPU を用いることを仮定した融合積和演算命令について述べる．融合積和演算命令は Intel AVX2 命令を搭載している Haswell マイクロアーキテクチャ以降で使用することができる．Intel AVX2 命令における主な融合積和演算命令を表 1 に示す [14].

ここでは，言語として C および C++を用いることを想定する．融合積和演算命令を適用する方法として，まず，コンパイラオプションによる指示が挙げられる．例えば，Intel C++コンパイラ [15] においては-march=core-avx2 を，GNU Complier Collection（GCC） [16] においては-mfma を指定すればよい．また，イントリンシック命令の利用が挙げられる．例えば，C および C++においては，まず，immintrin.h をインクルードする．そして，128 ビット・ベクトルの場合は__mm_fmadd_pd(__m128d x, __m128d y, __m128d z) を，256 ビット・ベクトルの場合は__mm256_fmadd_pd(__m256d x, __m256d y, __m256d z) を用いればよい．また，関数呼び出しによる適用が挙げられる．これは C99 標準ライブラリにおける math.h あるいは C++11 標準ライブラリにおける cmath で提供されている．

本稿では，融合積和演算命令の，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT への適用方法として，容易に利用することのできるコンパイラオプションによる指示を採用する．

4.5 半角公式を利用した三角関数テーブルの作成

FFT の実装で必要となる三角関数の計算は FFT ルーチン内で行うのではなく，あらかじめ計算してテーブルを作成し，参照による利用を行うことで効率を高くすることができる．浮動小数点演算による FFT で使用する三角関数の値を求める方法としては，C 言語における標準ライブラリ関数のようにあらかじめ用意されている関数を用いる方法がある．一方，上限-下限型区間演算矩形演算に基づく区間演算を用いた精度保証付き FFT および中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT で使用する区間表示された三角関数の値を求める方法としては，テイラー展開の利用や倍角公式の利用がある [17]．本稿では，入力データが 2 のべき乗の FFT の実装で必要となる三角関数の計算を容易に求めることのできる，半角公式を利用した計算方法を提案する．

Algorithm 8 半角の公式を利用した三角関数の計算（浮動小数点演算）

Input : $c_{m,n}$
Output : $c_{m,n}$
for $j = 1$ **to** n : $j = 2j$
  if $j = 1$ **then** $c_{m,n} \leftarrow 1$
  if $j \geq 2$ && $(m/j \,\%\, 2 = 0)$ **then**
    if $(m \,\%\, j) \,/\, j < \frac{1}{2}$ **then** $c_{m,n} \leftarrow \sqrt{\frac{1+c_{m,n}}{2}}$
    if $(m \,\%\, j) \,/\, j = \frac{1}{2}$ **then** $c_{m,n} \leftarrow 0$
    if $(m \,\%\, j) \,/\, j > \frac{1}{2}$ **then** $c_{m,n} \leftarrow -\sqrt{\frac{1+c_{m,n}}{2}}$
  if $j \geq 2$ && $(m/j \,\%\, 2 \neq 0)$ **then**
    if $(m \,\%\, j) \,/\, j < \frac{1}{2}$ **then** $c_{m,n} \leftarrow -\sqrt{\frac{1+c_{m,n}}{2}}$
    if $(m \,\%\, j) \,/\, j = \frac{1}{2}$ **then** $c_{m,n} \leftarrow 0$
    if $(m \,\%\, j) \,/\, j > \frac{1}{2}$ **then** $c_{m,n} \leftarrow \sqrt{\frac{1+c_{m,n}}{2}}$

4.5.1 浮動小数点演算を用いた三角関数テーブルの作成

浮動小数点演算による n 点 FFT では，$\omega_n^m = e^{-2\pi im/n} = \cos(-2\pi m/n) + i \cdot \sin(-2\pi m/n)$ を計算する必要がある．ただし，m は任意の自然数である．したがって，$\cos(-2\pi m/n)$ と $\sin(-2\pi m/n)$ の値を格納したテーブルを作成する必要があるが，実際は，式 (11) および式 (12) より，テーブルに $\cos(2\pi m/n)$ の値を格納しておけばよい．

$$\cos(-2\pi m/n) = \cos(2\pi m/n) \tag{11}$$

$$\sin\left(\frac{-2\pi m}{n}\right) = \cos\left(\frac{2\pi\left(\frac{n}{2}+m\right)}{n}\right) \tag{12}$$

ここで，n が 2 のべき乗である場合には，半角公式を繰り返し用いることで $\cos(2\pi m/n)$ の値を Algorithm 8 によって求めることができる．ただし，$c_{m,n}$ は $\cos(2\pi m/n)$ の値を格納する変数である．

4.5.2 上限-下限型区間演算を用いた三角関数テーブルの作成

上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT および中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT では，$\omega_n^m = e^{-2\pi im/n} = \cos(-2\pi m/n) + i \cdot \sin(-2\pi m/n)$ を厳密な値が包含されるように計算する必要がある．したがって，$\cos(-2\pi m/n)$ と $\sin(-2\pi m/n)$ の厳密な値を包含した区間を格納したテーブルを作成する必要があるが，実際は，4.5.1 節と同様の理由から，テーブルには $\cos(2\pi m/n)$ の値を格納しておけばよい．ただし，厳密な値を包含した区間は上限-下限型区間として表現する．ここで，IEEE754 における丸めの制御は四則演算と

Algorithm 9 半角の公式を用いた三角関数の計算（上下-下限型区間演算）

Input : $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right]$
Output : $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right]$
for $j = 1$ **to** $n : j = 2j$
  if $j = 1$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow [1, 1]$
  if $j \geq 2$ && $(m/j\ \%\ 2 = 0)$ **then**
    if $(m\ \%\ j)\ /\ j < \frac{1}{2}$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow \sqrt{\frac{[1,1]+\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right]}{[2,2]}}$
    if $(m\ \%\ j)\ /\ j = \frac{1}{2}$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow [0, 0]$
    if $(m\ \%\ j)\ /\ j > \frac{1}{2}$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow -\sqrt{\frac{[1,1]+\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right]}{[2,2]}}$
  if $j \geq 2$ && $(m/j\ \%\ 2 \neq 0)$ **then**
    if $(m\ \%\ j)\ /\ j < \frac{1}{2}$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow -\sqrt{\frac{[1,1]+\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right]}{[2,2]}}$
    if $(m\ \%\ j)\ /\ j = \frac{1}{2}$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow [0, 0]$
    if $(m\ \%\ j)\ /\ j > \frac{1}{2}$ **then** $\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right] \leftarrow \sqrt{\frac{[1,1]+\left[\underline{c_{m,n}}, \overline{c_{m,n}}\right]}{[2,2]}}$

平方根のみに適用されるため，$[\nabla\cos(2\pi m/n), \Delta\cos(2\pi m/n)]$ とすることはできない．そこで，三角関数の値は四則演算と平方根のみで求め，区間として表現する必要がある．ここで，n が 2 のべき乗である場合には，半角公式を繰り返し用いることで $\cos(2\pi m/n)$ の値を Algorithm 9 によって四則演算と平方根のみで求めることができる．ただし，四則演算と平方根は上限-下限型区間演算で行っていることに注意する．

ここで，Algorithm 9 で得られるのは上限-下限型区間であるが，中心-半径型区間演算に基づく矩形演算で用いるためには中心-半径型区間に変換する必要がある．その場合には，Algorithm 5 による変換を行う [1].

5. 性能評価

5.1 実験方法

入力データを一様乱数とし，$n = 2^m$ $(m = 8, 9, \cdots, 16)$ における浮動小数点演算による FFT，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間を測定し，比較する．ただし，FFT の実装において必要となる三角関数の計算は 4.5 節による方法であらかじめ行い，テーブルを作成して参照できるようにする．なお，三角関数のテーブルを作成する部分は経過時間の測定対象とはしない．また，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT，INTLAB で実装されている精度保証付き FFT において出力される結果の区間幅の最大値を測定する．

5.2 実験環境

浮動小数点演算による FFT，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT における経過時間の測定・最大区間幅の測定では，CPU は Intel(R) Xeon(R) CPU E5-2670 v3(2.30GHz) を用いた．また，コンパイラは Intel C++コンパイラのバージョン 15.0.2 を用いた．ビルド時のオプションは，処理速度を最大限に最適化する-O3 および融合積和演算命令を使用する-march=core-avx2 を指定した．中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の融合積和演算命令の使用はアセンブリコードにおける表 1 の VFMADD213PD，VFMADD231PD，VFMADD213SD，VFMADD231SD の融合積和演算命令によって確認した．中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT のベクトル化は-qopt-report=5 により出力される最適化レポートにより確認した．

INTLAB は Version9 を使用し，CPU は Intel Core i5-4288U (2.30GHz) を，MATLAB はバージョン R2014b を搭載した計算機上で最大区間幅の測定を行った．

5.3 実験結果

5.3.1 経過時間

浮動小数点演算による FFT，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間を図 3 に示す．ただし，横軸はデータ点数 n を，縦軸は経過時間（秒）を表す．実験結果より，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間は浮動小数点演算による FFT と比較して最小で約 13.1 倍（$n = 32768$），最大で約 22.4 倍（$n = 1024$）であった．一方，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間は浮動小数点演算による FFT と比較して最小で約 3.6 倍（$n = 32768$），最大で約 6.4 倍（$n = 1024$）であった．したがって，中心-半径型区間演算に基づく矩形演算による精度保証付き FFT の経過時間は上限-下限型区間演算に基づく矩形演算による精度保証付き FFT よりも高速であることが確認された．

ここで，補足実験として，浮動小数点演算による FFT，丸めモード変更回数の削減を行わない上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT，丸めモード変更回数の削減・融合積和演算命令の利用・ベクトル化を行わない中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間を測定した．補足実験の結果を図 4 に示す．実験結果より，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間は浮動小数点演算による FFT と比較して最小で約 33 倍（$n = 32768$），最大で約 58.7 倍（$n = 1024$）であった．一方，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間は浮動小数点演算による FFT と比較して最小で約 30.5 倍（$n = 32768$），最大で約 54.8 倍（$n = 1024$）であった．補足実験の結果から，上限-下限型区間演算に基づく矩

形演算を用いた精度保証付き FFT は丸めモード変更回数の削減によって高速化につながったことが分かる．また，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT は丸めモード変更回数の削減・融合積和演算命令の利用・ベクトル化のいずれかによって高速化につながったことが分かる．

さらに，追加の補足実験として，融合積和演算の命令の利用のみ行わない中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間を測定した．なお，浮動小数点演算による FFT，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT は最初の実験と同様である．追加の補足実験の結果を図 5 に示す．実験結果より，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT は浮動小数点演算による FFT と比較して最小で約 4.1 倍（$n = 65536$），最大で約 9 倍（$n = 512$）であった．また，ベクトル化のみ行わない中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間を測定した．なお，浮動小数点演算による FFT，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT は最初の実験と同様である．実験の結果を図 6 に示す．実験結果より，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT は浮動小数点演算による FFT と比較して平均で最小で約 3.8 倍（$n = 32768$），最大で約 6.8 倍（$n = 1024$）であった．以上の追加実験により，融合積和演算命令を利用しない，または，ベクトル化を行わないことによる中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の経過時間は多少の変化は見られたものの，図 4 のような変化は起こらないことが分かる．したがって，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT は丸めモード変更回数の削減によって特に高速化につながったことが分かる．なお，融合積和演算命令の利用が高速化に大きな影響を与えるまでには至らなかった理由として，VFMADD213PD，VFMADD231PD に加えて VFMADD213SD，VFMADD231SD の命令によって計算を行っている箇所があることや，融合積和演算命令を 1 回適用すれば済む $x \times y + z$ の形をした式が FFT ルーチンの最内側ループに無いことが原因として考えられる．また，ベクトル化の利用が高速化に大きな影響を与えるにまでは至らなかった理由として，FFT ルーチンの最内側ループで使用している変数を配列に変更する箇所が生じたことが原因として考えられる．

5.3.2 最大区間幅

上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT，INTLAB による精度保証付き FFT の最大区間幅を表 2 に示す．

表 2 より，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の最大区間幅は，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT の最大区間幅よりも大きくなることが分かる．これは，乗算における，中心-半径型区間演算の上限-下限型区間演算に対する過大評価が原因と考えられる．しかし，5.3.1 節で示したように，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT は上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT よりも高速に計算できる点で有用である．

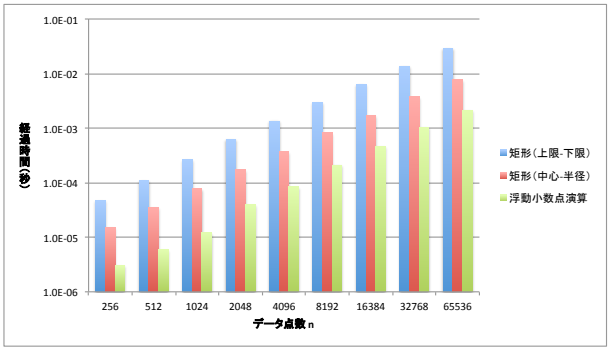

図 3 各 FFT の経過時間

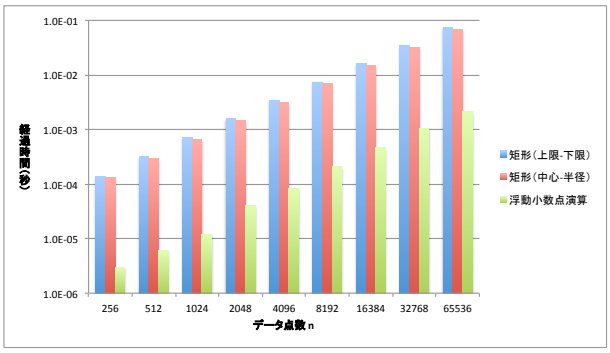

図 4 各 FFT の経過時間（高速化前）

また，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT と中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の両者の最大区間幅は INTLAB による精度保証付き FFT の最大区間幅に対し，データ点数が小さいときは最大区間幅が小さく，データ点数が大きくなるに従って最大区間幅が大きくなることが分かる．これは，INTLAB による精度保証付き FFT が円板演算による精度保証付き FFT であるからと考えられる．実際，3.3 節で述べたように，野村の研究において，データ点数が小さいとき，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT は円板演算を用いた精度保証付き FFT よりも最大区間幅が小さく，データ点数が大きいとき，円板演算を用いた精度保証付き FFT は上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT よりも区間幅が大きいという結果を示した [5].

6. まとめ

本稿では，既存の上限-下限型区間演算に基づく矩形演算

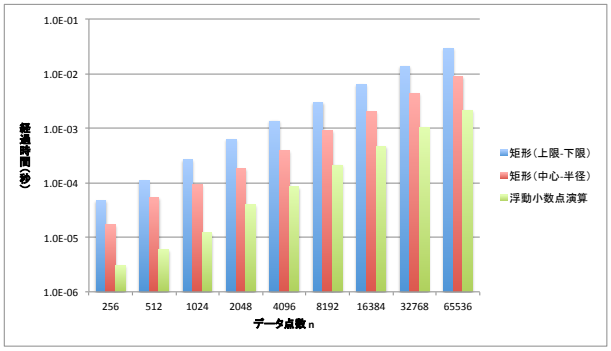

図 5 各 FFT の経過時間（融合積和演算命令未使用）

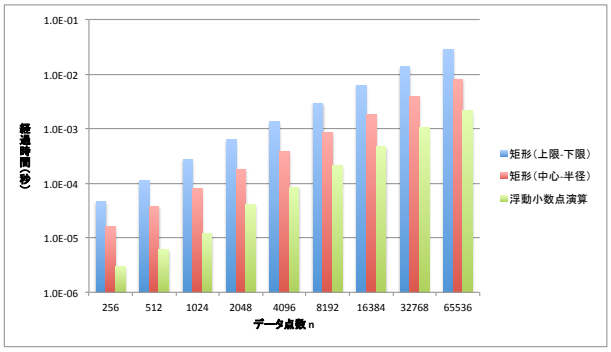

図 6 各 FFT の経過時間（非ベクトル化）

表 2 最大区間幅

n	矩形（上限-下限）	矩形（中心-半径）	INTLAB
256	1.6165×10^{-13}	3.0337×10^{-13}	5.1159×10^{-13}
512	4.5475×10^{-13}	7.9085×10^{-13}	1.0232×10^{-12}
1024	1.2363×10^{-12}	2.1042×10^{-12}	2.5011×10^{-12}
2048	3.0376×10^{-12}	5.1943×10^{-12}	5.0022×10^{-12}
4096	7.7094×10^{-12}	1.3118×10^{-11}	1.0914×10^{-11}
8192	1.9462×10^{-11}	3.3112×10^{-11}	2.0918×10^{-11}
16384	4.9852×10^{-11}	8.4702×10^{-11}	5.0932×10^{-11}
32768	1.2613×10^{-10}	2.1447×10^{-10}	8.0036×10^{-11}
65536	3.2097×10^{-10}	5.4832×10^{-10}	1.8917×10^{-10}

を用いた精度保証付き FFT よりも高速に計算することを目的とし，新たに中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT を提案した．さらに，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT と中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT で丸めモード変更の回数を削減できることを示した．また，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT において融合積和演算命令の利用とベクトル化を行った．一方，半角公式を用いた三角関数の新たな計算方法も示した．

そして，性能評価の結果から，浮動小数点演算を用いた FFT と比較して，上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT の計算時間は最小で約 13.1 倍，最大で約 22.4 倍となり，一方，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT の計算時間は最小で約 3.6 倍，最大で約 6.4 倍となり，中心-半径型区間演算に基づく矩形演算を用いた精度保証付き FFT が上限-下限型区間演算に基づく矩形演算を用いた精度保証付き FFT よりも高速であることを確認した．

今後の課題として，ロード/ストアの回数を減らすため，2 以外の基数における精度保証付き FFT の実装を行うことが挙げられる．また，入力のデータ数が 2 のべき乗以外の精度保証付き FFT を実現し，汎用性を高めることが挙げられる．

謝辞 本研究の一部は，JST CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究領域の支援によって行われた．

参考文献

[1] 大石進一：精度保証付き数値計算，コロナ社 (2000).
[2] 山中脩也：高信頼・高精度・高可搬な数値計算法の研究，博士論文，早稲田大学 (2011).
[3] 近藤祐史，野田松太郎：数式処理と区間演算の結合－複素区間数の場合－，数理解析研究所講究録，No. 848, pp. 132–146 (1993).
[4] Cooley, J. W. and Tukey, J. W.: An Algorithm for the Machine Calculation of Complex Fourier Series, *Mathematics of Computation*, Vol. 19, pp. 297–301 (1965).
[5] 野村久美子：精度保証付き高速フーリエ変換，卒業論文要旨, 南山大学 (2009).
[6] Wolfram: WOLFLAM MATHEMATICA, http://www.wolfram.com/mathematica/.
[7] MathWorks: MATLAB, http://www.mathworks.com/products/matlab/.
[8] Eaton, J. W.: GNU Octave, https://www.gnu.org/software/octave/.
[9] Rump, S. M.: INTLAB - INTerval LABoratory, http://www.ti3.tu-harburg.de/rump/intlab/.
[10] Chonali, N. D.: Implementation Of Eight Point FFT Algorithm Using Interval Arithmetic On FPGA, *International Journal of Emerging Technology in Computer Science and Electronics*, Vol. 14, No. 2, pp. 1–4 (2015).
[11] Rump, S. M.: Fast And Parallel Interval Arithmetic, *BIT Numerical Mathematics*, Vol. 39, No. 3, pp. 539–560 (1999).
[12] Bailey, D. H.: A High-Performance FFT Algorithm for Vector Supercomputers, *International Journal of Supercomputer Applications*, Vol. 2, pp. 82–87 (1988).
[13] Swarztrauber, P. N.: FFT Algorithms for Vector Computers, *Parallel Comput.*, Vol. 1, No. 1, pp. 45–63 (1984).
[14] Intel: Intel 64 and IA-32 Architectures Software Developer's Manual, http://www.intel.co.jp/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf.
[15] Intel: Intel C++ Compilers, https://software.intel.com/en-us/c-compilers.
[16] Free Software Foundation: GNU Compiler Collection, https://gcc.gnu.org/.
[17] 西原伸也，宮田孝富，柏木雅英：倍角公式を用いた三角関数の精度保証，2002 年電子情報通信学会基礎・境界ソサイエティ大会，p. 55 (2002).